A Declarative Paradigm for Robust Cumulative Scheduling

Alban Derrien, Thierry Petit, Stéphane Zampelli

A Declarative Paradigm for Robust Cumulative Scheduling

Alban Derrien, Thierry Petit and Stéphane Zampelli

TASC (Mines Nantes, LINA, CNRS, INRIA),
4, Rue Alfred Kastler, FR-44307 Nantes Cedex 3, France.
{alban.derrien, thierry.petit}@mines-nantes.fr, szampelli@gmail.com

Abstract. This paper investigates cumulative scheduling in uncertain environments, using constraint programming. We present a new declarative characterization of robustness, which preserves solution quality. We highlight the significance of our framework on a crane assignment problem with business constraints.

1 Introduction

Scheduling consists in assigning activities over time. When a solution is executed in a real-world environment, activities may take longer to execute than expected. In many practical cases, solutions cannot be re-computed at anytime when disruptions occur. For instance, in Crane Assignment [17], planners need a fixed schedule which guarantees that the vessel processing will be completed ahead of schedule. The solution should meet the deadline while being able to absorb activity delays during its execution. We wish a tradeoff between robustness and performance.

We aim to address this issue for the Cumulative Scheduling Problem (CuSP), with possibly additional constraints. In a CuSP, each activity $a \in \mathcal{A}$ has a starting time variable s_a and an ending time variable e_a. Its duration p_a (processing time) and resource consumption h_a are usually strictly positive integers. We use the notation $a = \langle s_a, p_a, e_a, h_a \rangle$. Given an integer capacity C, a solution to a CuSP satisfies the following constraints: $\forall a \in \mathcal{A}, s_a + p_a = e_a$ and $\forall t \in \mathbb{N}, \sum_{t \in [s_a, e_a[, a \in \mathcal{A}} h_a \leq C$. In Constraint Programming, the *Cumulative*$(\mathcal{A}, C)$ constraint [3] represents a CuSP. A usual objective is to minimize the *makespan*, i.e., the latest end among all activities.

In this paper, we integrate the notion of robustness directly into the problem definition. We define a new generic problem, such that any activity can be delayed up to a certain time without being forced to re-schedule the other activities in its neighborhood. We introduce *FlexC*, a constraint dedicated to this problem. Our paradigm deals with the three following aspects at the same time. 1. *Declarative framework*: We solve problems such that solutions cannot be recomputed at anytime. Maximum allowed delays of activities are a data and may vary from one activity to another. 2. *Specialized definition*: In order to fit the practical needs, we use a robustness definition based on the semantics of the core problem. Notably, all the variables have not the same status. 3. *Modular approach*: We design a framework such that the model should not be totally re-written each time we add a new business constraint.

Crane Assignment is a real-world example with these three requirements. In order to validate our approach, we experiment on this problem.

2 Robust Cumulative Scheduling

Related Work In the literature, some frameworks deal with the three aspects (*declarative approach*, *specialized definition* and *modularity*), but not all at the same time for the CuSP. In a super-solution [9; 10], the loss of the values of at most a variables can be repaired by changing the values of these variables and at most b other variables. This notion is generic. All variables have the same status. It has been applied to job-shop benchmarks [9]. A low-performance technique for obtaining robust schedules is to augment the duration of each activity. To improve it, some slack-based techniques incorporate the reasoning about uncertainty in the solving process [7]. This approach does not deal with CuSP and its modularity was not investigated but it has some links with our work: Schedules absorb some level of unexpected events without rescheduling. Other Operations Research techniques for robustness in scheduling problems different from the CuSP can be found in [5; 16]. A Mixed-Integer Linear Programming formulation of robust RCPSP (i.e., CuSP with precedences) has been proposed in [14]. This formulation requires an exponential number of variables and constraints.

A New Framework for Robust CuSP We use the following notation for i-order maximum heights of activities: Given $\mathcal{A}^{\downarrow}$ the collection of activities in a set $\mathcal{A}$ sorted by decreasing heights, $max^i_{a\in\mathcal{A}}(h_a)$ is the height of the i^{th} activity in $\mathcal{A}^{\downarrow}$.

We propose a new definition of cumulative problems, where each activity a can be delayed up to k_a points in time, without modifying the position of any other activity in its neighborhood. This can be viewed as a specialization of the notion of super-solutions that takes into account the semantics of the CuSP. Formally, given an integer $r \geq 1$, we define the Robust Cumulative Problem of order r (RCuSP^r).

Definition 1 (RCuSP^r). *Given a set of activities $\mathcal{A}$, let $\mathcal{K}$ be a set of positive integers slacks associated with activities, such that to each $a \in \mathcal{A}$ corresponds $k_a \in \mathcal{K}$. Let r be an integer, $r \geq 1$. A solution to a* RCuSP^r *satisfies the following constraints:*

$$\forall a \in \mathcal{A}, s_a + p_a = e_a \quad \wedge \quad \forall t \in \mathbb{N}, \sum_{\substack{a\in\mathcal{A},\\ t\in[s_a,e_a[}} h_a + \sum_{i=1}^{i=r} \underset{a\in\{b\in\mathcal{A},t\in[e_b,e_b+k_b[\}}{max^i}(h_a) \leq C$$

Definition 1 considers a set $\mathcal{K}$ of positive integers for slacks. The material presented in this paper is consistent with the case where $\mathcal{K}$ is a set of variables, provided that values in their domains are greater than or equal to 0 and for each activity a we consider k_a as the minimum valid value for the variable in $\mathcal{K}$ mapped with a. Using variables, constraints can be defined on slacks, e.g., dependencies on starting time of activities.

We now focus on the problem RCuSP (RCuSP^r with $r = 1$). We express a RCuSP with a constraint, *FlexC*$(\mathcal{A}, C, \mathcal{K})$, as it is done for the CuSP.

Property 1. *FlexC*$(\mathcal{A}, C, \mathcal{K}) \Rightarrow$ *Cumulative*$(\mathcal{A}, C)$.

Proof. By Definition 1 and Definition of the CuSP. □

Property 2. Assume activities in $\mathcal{A}$ are fixed. Let $a = \langle s_a, p_a, e_a, h_a\rangle$ be an activity in $\mathcal{A}$, and k an integer, $0 \leq k \leq k_a$. Given $a' = \langle s'_a = (s_a + k), p_a, e'_a = (e_a + k), h'_a = h_a\rangle$ and $\mathcal{A}' = \mathcal{A} \cup \{a'\} \setminus \{a\}$, we have: *FlexC*$(\mathcal{A}, C, \mathcal{K}) \Rightarrow$ *Cumulative*$(\mathcal{A}', C)$.

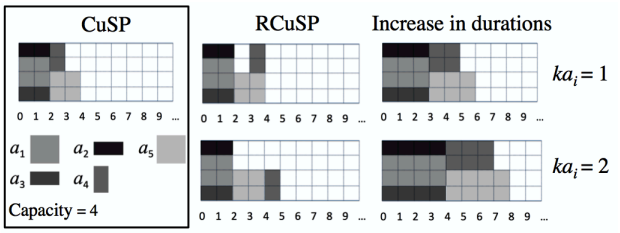

Fig. 1. Solutions minimizing the makespan. Grey rectangles are activities, the horizontal axis is time and the vertical one is consumption. On the left, a CuSP without robustness. In the middle, optimal solutions of the RCuSP with values in $\mathcal{K}$ respectively all equal to 1 (top) and 2 (bottom). On the right, optimal solutions with all durations increased respectively by 1 and 2.

Proof. Assume $\neg$ *Cumulative*$(\mathcal{A}', C) \wedge$ *FlexC*$(\mathcal{A}, C, \mathcal{K})$ (hypothesis). As $\neg$ *Cumulative*$(\mathcal{A}', C)$, $\exists t_{fail} \in \mathbb{N}$, $\sum_{b \in \mathcal{A}', t_{fail} \in [s_b, e_b[} h_b > C$ (1). From Property 1, *FlexC*$(\mathcal{A}, C, \mathcal{K}) \Rightarrow$ *Cumulative*$(\mathcal{A}, C)$. Then, $\forall t \in \mathbb{N}, \sum_{b \in \mathcal{A}, t \in [s_b, e_b[} h_b \leq C$. As $\mathcal{A}$ and $\mathcal{A}'$ differ only wrt. a and a', $t_{fail} \in [\max(e_a, s'_a), e'_a[$. As *FlexC*$(\mathcal{A}, C, \mathcal{K})$ is satisfied, $(\sum_{b \in \mathcal{A}, t_{fail} \in [s_b, e_b[} h_b) + (\max_{b \in \mathcal{A}, t_{fail} \in [e_b, e_b + k_b[} h_b) \leq C$ (2). (2) minus (1) leads to $(\max_{b \in \mathcal{A}, t_{fail} \in [e_b, e_b + k_b[} h_b) - h'_a < 0$. As $t_{fail} \in [\max(e_a, s'_a), e'_a[$, $h_a \leq (\max_{b \in \mathcal{A}, t_{fail} \in [e_b, e_b + k_b[} h_b)$. Thus, $h_a - h'_a < 0$, absurd by definition of $\mathcal{A}'$. □

Delaying the starting time of an activity or increasing its duration are equivalent in the context of the RCuSP, provided we do not both delay and enlarge the activity. In case of enlargement ($a' = \langle s_a, p'_a = (p_a + k), e'_a = (e_a + k), h'_a = h_a \rangle$), the proof of Property 2 is the same. Given any set of activities, comparing RCuSP^r when r varies, we can say that RCuSP^r has an optimum makespan less than or equal to the minimum makespan of RCuSP^{r+1}. When r is too high (e.g., with sum_r the sum of the r minimum heights of distinct activities, $sum_r \geq C$) we obtain the naive approach consisting in adding k_a to the duration of each activity. This naive method is the least performant and the most robust, as Fig. 1 shows. Conversely, the RCuSP (when r=1) is the most performant tradeoff. Nevertheless, we claim that the level of robustness in solutions of a RCuSP will be widely satisfactory in most of cases. Indeed, the RCuSP allows to delay several activities in a solution provided they are scheduled in disjoint intervals in time even when they are delayed. This property is the key of the practical significance of the RCuSP. Furthermore, in some particular solutions of *FlexC*, more than one activity (e.g., a_2 and a_3 in the medium picture of Fig. 1, with all values in $\mathcal{K}$ equal to 1) can be delayed in the same time window without violating *Cumulative*. Regarding modularity, as we define a constraint, our declarative paradigm can be combined with other constraints. To guarantee the robustness, additional constraints may also have to be modified. We demonstrate this possibility in Sect. 4.

3 Filtering Technique

This Sect. presents a Time-Table filtering for *FlexC*. We selected this algorithm because it is the best one in terms of scaling (number of activities) in the CuSP case [13]. This algorithm does not directly depends on the selected time unit. Given a variable x, $\underline{x}$ (resp. $\overline{x}$) denotes the minimum value (resp. the maximum value) in its domain.

Time-Table Failure Detection. We first study the failure condition of the Time-Table filtering Algorithm for *Cumulative*, such as Letort et al.'s algorithm [13]. It is based on the profile of compulsory parts [11]. The compulsory part of an activity $a \in \mathcal{A}$ is the interval in time where a has to be processed. This interval is $[\overline{s_a}, \underline{e_a}[$ (empty if $\overline{s_a} \geq \underline{e_a}$). The *profile* is the cumulated sum of heights of compulsory parts for each point in time t, which should never exceed the capacity C. From [2], we have:

Proposition 1 (Time-Table failure check for *Cumulative*).
If $\exists t \in \mathbb{N}, (\sum_{a \in \mathcal{A}, t \in [\overline{s_a}, \underline{e_a}[} h_a) > C$, *Cumulative*$(\mathcal{A}, C)$ *has no solution.*

To provide a similar failure condition for ***FlexC***, we have to add the necessary height to preserve the robustness (array $\mathcal{K}$) to the cumulated sum of heights of activities having a non-empty compulsory part. To do so, we introduce $\mathcal{K}$-compulsory parts. Given the compulsory part I_a of an activity a computed with the hypothesis that duration of a is $p_a + k_a$, the $\mathcal{K}$-compulsory part of a is the sub-interval of I_a that is not intersecting the initial compulsory part of a, $[\overline{s_a}, \underline{e_a}[$, if such a sub-interval exists (it can be empty).

Definition 2 ($\mathcal{K}$-compulsory part). *Let $a \in \mathcal{A}$ be an activity and $k_a \in \mathcal{K}$. The* $\mathcal{K}$-compulsory part *of a, denoted KCP_a, is the interval* $[\max(\overline{s_a}, \underline{e_a}), \underline{e_a} + k_a]$.

The Time-Table failure condition of ***FlexC*** integrates in the profile, at any time t, the maximum height among activities having a $\mathcal{K}$-compulsory part intersecting t.

Proposition 2 (Time-Table failure check for *FlexC*).
If $\exists t \in \mathbb{N}, (\sum_{a \in \mathcal{A}, t \in [\overline{s_a}, \underline{e_a}[} h_a) + (\max_{a \in \mathcal{A}, t \in KCP_a} h_a) > C$ *then* ***FlexC***$(\mathcal{A}, C, \mathcal{K})$ *fails.*

Proof. Assume $\exists t, \sum_{a \in \mathcal{A}, t \in [\overline{s_a}, \underline{e_a}[} h_a + \max_{a \in \mathcal{A}, t \in KCP_a} h_a > C$. Let b be an activity such that $t \in KCP_b$ and $h_b = \max_{a \in \mathcal{A}, t \in KCP_a} h_a$. Consider $\mathcal{A}' = \mathcal{A} \setminus \{b\} \cup \{b' = \langle s'_b = s_b, p'_b = p_b + k_b, e'_b = e_b + k_b, h_b \rangle\}$. By construction $t \in [\overline{s_{b'}}, \underline{e_{b'}}[$. $\sum_{a \in \mathcal{A}', t \in [\overline{s_a}, \underline{e_a}[} h_a > C$. *Cumulative*$(\mathcal{A}', C)$ is violated. By Property 2, ***FlexC***$(\mathcal{A}, C, \mathcal{K})$ is violated. □

Pruning Characterization. We assume now that, for each activity $a \in \mathcal{A}$, the solver maintains Bounds-Consistency [4] (BC) on the constraint $s_a + p_a = e_a$, independently from our propagator. A special case of ***FlexC***$(\mathcal{A}, C, \mathcal{K})$ is the case where all values in K are equal to 0. In this case, from Definition 1, ***FlexC*** $\Leftrightarrow$ *Cumulative*. As enforcing BC for *Cumulative* is NP-Hard [1], it is NP-Hard for ***FlexC***. Therefore, we consider a weaker form of BC. Our goal is that the achieved consistency corresponds to the filtering enforced by Time-Table in the case of *Cumulative*.

Definition 3. *Given a scheduling constraint, a propagator is* Time-Table *if $\forall a \in \mathcal{A}$, fixing s_a at time $\underline{s_a}$ (respectively, e_a at time $\overline{e_a}$) does not lead to a contradiction if we apply the Time-Table Failure check of the constraint.*

Fix-point property. The following property holds when Letort et al.'s *sweep_min* algorithm reaches its fixpoint (Property 1 in [13]) on lower bounds of start variables.

Property 3. Given *Cumulative*$(\mathcal{A}, C)$, *sweep_min* ensures that: $\forall b \in \mathcal{A}, \forall t \in [\underline{s_b}, \underline{e_b}[, h_b + \sum_{a \in \mathcal{A} \setminus \{b\}, t \in [\overline{s_a}, \underline{e_a}[} h_a \leq C$.

To adapt the fixpoint Property 3 to the case of *FlexC*, we have to ensure that any activity $b \in \mathcal{A}$ could be able to be scheduled at its earliest time $\underline{s_b}$ without leading directly to a fail when we apply Prop. 2.

$$\forall t \in [\underline{s_b}, \underline{e_b}[, (h_b + \sum_{\substack{a \in \mathcal{A} \setminus \{b\}, \\ t \in [\overline{s_a}, \underline{e_a}[}} h_a) + (\max_{\substack{a \in \mathcal{A}, \\ t \in KCP_a}} h_a) \leq C$$

This condition guarantees that when all variables are instantiated we have a solution of *FlexC*. The obtained filtering is weaker than Time-Table. For instance, consider a capacity $C = 1$ and two activities a_1 and a_2, such that a_2 is fixed to $\langle s_{a_2} = 4, p_{a_2} = 2, e_{a_2} = 6, h_{a_2} = 1 \rangle$, with $k_{a_2} = 0$. Assume $s_{a_1} = [0, 1000]$, $p_{a_1} = 2$, $h_{a_1} = 1$ and $k_{a_1} = 3$. The lower bound $\underline{s_{a_1}} = 0$ satisfies the previous condition. However, scheduling a_1 at $s_{a_1} = 0$ leads to a fail using Prop. 2. The condition ensures the consistency of each activity a all along its duration if scheduled at $\underline{s_a}$, but it does not guarantee that the space required after a to make it robust does not induce an inconsistency (because some activities may end after $\underline{e_a}$). The complete Time-Table fixpoint conditions are the following. Any activity which would lead to a Time-Table fail if fixed at its earliest (resp. latest) date violates one of the conditions, and reciprocally.

Property 4 (FlexC (lower bounds)). Given *FlexC*$(\mathcal{A}, C, \mathcal{K})$, the propagator should ensure $\forall b \in \mathcal{A}$:

$$\forall t \in [\underline{s_b}, \underline{e_b}[, (h_b + \sum_{\substack{a \in \mathcal{A} \setminus \{b\}, \\ t \in [\overline{s_a}, \underline{e_a}[}} h_a) + (\max_{\substack{a \in \mathcal{A}, \\ t \in KCP_a}} h_a) \leq C \quad (1)$$

$$\wedge \forall t \in [\underline{e_b}, \underline{e_b} + k_b[, (\sum_{\substack{a \in \mathcal{A}, \\ t \in [\overline{s_a}, \underline{e_a}[}} h_a) + h_b \leq C \quad (2)$$

Property 5 (FlexC (upper bounds)). Given *FlexC*$(\mathcal{A}, C, \mathcal{K})$, the propagator should ensure the same conditions as Property 4 with intervals $[\overline{s_b}, \overline{e_b}[$ (condition (1)) and $[\overline{e_b}, \overline{e_b} + k_b[$ (condition (2)).

We can obtain this filtering using either a decomposition or a dedicated algorithm.

Decomposition. Let $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ be a set of activities. For each $a_i = \langle s_{a_i}, p_{a_i}, e_{a_i}, h_{a_i} \rangle$ in $\mathcal{A}$ we define $a'_i = \langle s_{a_i}, p_{a'_i} = (p_{a_i} + k_{a_i}), e_{a'_i} = (e_{a_i} + k_{a_i}), h_{a_i} \rangle$ and $\mathcal{A}_i = \mathcal{A} \cup \{a'_i\} \setminus \{a_i\}$. The set of solutions of *FlexC*$(\mathcal{A}, C, \mathcal{K})$ is the set obtained by projecting on variables in $\mathcal{A}$ all solutions of the following constraint network $\mathcal{CN}$: *Cumulative*$(\mathcal{A}_1, C, \mathcal{K}) \wedge$ *Cumulative*$(\mathcal{A}_2, C, \mathcal{K}) \wedge \ldots \wedge$ *Cumulative*$(\mathcal{A}_n, C, \mathcal{K})$. Representing a global constraint with n global constraints may be costly. With respect to R$_r$CuSP, $\binom{n}{r}$ *Cumulative* constraints are required. However, using Time-Table for each *Cumulative* of the decompositon prunes the same values as Time-Table for *FlexC*.

Dynamic Sweep Time-Table algorithm. We have adapted [8] the Time-Table Letort et al.'s dynamic Sweep algorithm [13; 12] for *Cumulative*, in order to design a propagator for *FlexC*. This algorithm is in two steps: Filtering of lower bounds of starting time variables (*Sweep_min*) and upper bounds of ending-time variables (*Sweep_max*). *Sweep_min* for *FlexC* is in $O(n^2)$ time, as for *Cumulative* [12, p. 55]. Conversely to

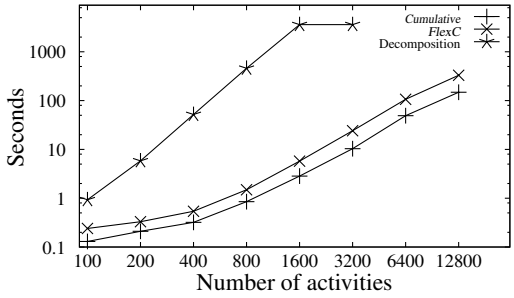

Fig. 2. Scaling of Dynamic Sweep for *FlexC*.

the case of *Cumulative*, the filtering of *FlexC* is not symmetrical. In *Sweep_max*, our implementation adds new events in the sweep process to handle $\mathcal{K}$-compulsory parts, leading to a $O(n^2 \times \max_{a \in \mathcal{A}}(k_a)))$ time algorithm. As there is some differences with Sweep for *Cumulative*, we have experimented the limits of our algorithm with respect to problems size. We used Choco [6] with a 2.9 Ghz Intel i7 and 8GB of RAM. Following experiments provided in [12], we generated large random instances with p_a from 5 to 10, h_a from 1 to 5, $C = 30$. Values in $\mathcal{K}$ are not null, with an average equal to 4. Similar results are obtained with fixed k_a. Figure 2 shows that our filtering algorithm scales on problems up 12800 activities for a first solution. The decomposition reaches the time limit of 1h:00m with 1600 activities and leads to a memory crash with 6400 *Cumulative*.

4 Experiments with Side Constraints

FlexC can be used in a closed world, but external constraints can also be defined. It may be necessary to make them robust. For instance, precedence constraints of the form $e_{a_i} + \Delta_{ij} \leq s_{a_j}$. As the constraint $\leq$ is monotone, using the natural ordering of integers, augmenting s_{a_j} by k_{a_j} does not reduce the set of solutions of $e_{a_i} + \Delta_{ij} \leq s_{a_j}$. Conversely, to ensure that solutions are robust to the increase of e_{a_i}, the precedences should be strengthened: $e_{a_i} + k_{a_i} + \Delta_{ij} \leq s_{a_j}$. The principle can be extended to more complex constraints, e.g., business constraints of the Crane Assignment Problem (CAP).

We now present experiments on this problem. Our goal is to show how the model with business side constraints (such as precedence constraints, transition times, machine assignments) can be transformed into a robust one, and to measure the impact of the robust model on the objective function compared to a naive model where all durations are extended. The CAP is a specialization of the berth and crane problem [17], where we focus on the detailed scheduling of a single-container cargo's discharge. A cargo vessel is made of bays. Bays are transverse sections storing containers. Each bay is split into above deck and below deck parts. Below and above bays hold containers. A fixed number of cranes is assigned to the vessel. Cranes are operated on a single rail, they cannot cross each other. The goal is to minimize the makespan: The terminal has to pay a fee proportional to the leaving time of the cargo. Crane productivity depends on the wind and sea conditions, on the driver, and on the position of the containers in the cargo. To avoid fees, customers wish a fixed "worst case" schedule which guarantees to meet

```
range A=1..40//Range of acts                                  1
int nbc=4; //nbr of cranes                                    2
int pos[A]=...; //act position                                3
int tt[A,A]=...;//acts transition time                        4
bool preced[A,A]=...;//act precedence                         5
Solver m();                                                   6
IntVar s[A](...);//start                                      7
IntVar p[A](m,rand[5,maxd]));//duration                       8
IntVar e[A](...);//end                                        9
IntVar h[A](m,1);//resource                                   10
IntVar c[A](m,[0,nbc-1]);//crane                              11
IntVar k[i∈A](m,rand([0,.25])*p[i]);                          12
```

Fig. 3. CAP Input Data.

```
//1.cumu cstr, nbc resources                                  1
m.post(Cumulative(s,p,e,h,nbc))                               2
//2.precedence constraints                                    3
for (i,j) s.t. preced[i,j]==1:                                4
   m.post(e[i]<s[j]);                                         5
//3.crane alloc, transition times                             6
for (i,j) i!=j∧pos[i]<pos[j]:                                 7
   m.post( ((s[i]<e[j]+tt[i,j])                               8
      ∧ (s[j]<e[i]+tt[i,j]))                                  9
         => c[i]<c[j] );                                      10
//4.no intersec for nearby acts                               11
for (i,j) i<j∧|pos[i]-pos[j]|<=2:                             12
   m.post(s[i]>e[j] ∨ e[i]<s[j]);                             13
minimize obj = min({e[i]} i∈ A)                               14
```

Fig. 4. CAP model.

```
//1.flex cumu cstr, nbc resources                             1
m.post(FlexC(s,p,e,h,k,nbc))                                  2
//2.precedence constraints                                    3
for (i,j) s.t. preced[i,j]==1:                                4
   m.post(e[i]+k[i] <s[j]);                                   5
//3.crane alloc, transition times                             6
for (i,j) i!=j∧pos[i]<pos[j]:                                 7
   m.post( ((s[i]<e[j]+tt[i,j]+k[j])                          8
      ∧ (s[j]<e[i]+tt[i,j]+k[i]))                             9
         => c[i]<c[j] );                                      10
//4.no intersec for nearby acts                               11
for (i,j) i<j∧|pos[i]-pos[j]|<=2:                             12
   m.post(s[i]>e[j]+k[i] ∨ e[i]+k[i]<s[j]);                   13
minimize obj = min({e[i]+k[i]} i∈ A)                          14
```

Fig. 5. Robust CAP model.

a deadline, given a precise robustness definition, as our framework does. Simulation is not relevant: Uncertainty has to be taken into account *a priori* in the problem definition. The maximum allowed slack k_a for each activity a is a data. Generating durations a posteriori is not relevant, as they would have to match the input robustness criteria.

Data. Figure 3 provides the pseudo code for the input data and decision variables. Line 1 is the range of activities. The cargo has 20 bays with one activity below and above deck. Line 2 sets the number of resources to 4, the typical number of cranes for such a cargo of 20 bays. Lines 3-5 declare the bay position of each activity, the transition time and the presence of a precedence constraint. Transition times are computed based on the distance between two positions, multiplied by a factor. For each bay, we have a precedence constraint between below and above deck activities. An additional 5% of precedences are randomly set to reflect discharge balance constraints on the cargo. Lines 7-10 declare the start, duration, end, and resource variables for each activity. Durations (in minutes) are randomly chosen in $[5, \text{maxd} = 800]$ to capture many scenarios. Value 800 is the maximum duration for discharging a large bay. In lines 11-12, additional crane and robustness variables are created for each activity. The robust factor k, fixed, is randomly chosen in $[0, 25\%]$ multiplied by the duration of the activity. This is a bad case for our approach (performance improves as the ratio k_a/p_a increases).

Constraints. Figure 4 shows the constraints of the CAP model without robustness. Following [17], we model this application as a cumulative scheduling problem. Lines 2-5 post the cumulative constraint and precedence constraints. Lines 8-10 post the crane allocation constraints. Given two different activities i and j with i being on the left of j, if those activities intersect in time, we ensure that crane assignment follows their position, assuming crane 0 is on the left of crane 1. Those constraints ensure that any set of activities intersecting with a given point in time has a feasible crane assignment. The activities should not be assigned to the same crane if they intersect in time while being extended by the transition time, because a crane would not have the required time to travel from activity i to activity j. Line 13 ensures that, for security reasons, cranes

should not work on nearby bays. If two activities are two bays away from each other, they should have no intersection in time.

Robust Model. In Fig. 5, *FlexC* is used with the k variables in line 2. The precedences constraints are modified in line 5 to accomodate the robust factor. The left side of the constraint in lines 8-10 is an intersection in time condition. The solution should ensure that if an activity is pushed or extended and intersects after this change with another activity, the schedule is still valid. Line 13 posts the negation of an intersection in time condition, and we can add $k[i]$ to $e[i]$.

Heuristic. For the three models, first the starting variables are assigned based on min domain and min value, then on crane variables. The search strategy uses propagation-guided LNS [15] and fixes randomly 70% of the starting time of the activities.

#	CAP	*FlexC*	Naive	α	γ
1	10.2 (0.08)	20.7 (2.20)	36.2 (3.28)	0.40	2.47
2	8.6 (0)	19.4 (2.13)	32.5 (1.62)	0.45	2.21
3	5.7 (0)	19.2 (3.08)	28.5 (0.60)	0.59	1.68
4	9.3 (0)	17.7 (0.47)	30.9 (3.66)	0.38	2.57
5	6.3 (0)	20.2 (2.10)	35.3 (0)	0.47	2.08
6	6.4 (0)	17.8 (1.37)	30.7 (0.15)	0.46	2.13
7	4 (0)	15.5 (1.08)	35.8 (2.40)	0.36	2.76
8	1.8 (0)	19.8 (2.35)	27.1 (1.42)	0.71	1.40
9	13.2 (0)	15.5 (2.61)	22 (0.31)	0.26	3.82
10	7.2 (0)	19.4 (1.63)	31.6 (0.07)	0.5	2.0

Table 1. Lower bound distance in %.

Performance. We measure the distance of the objective value to a lower bound computed by adding the durations and dividing by the number of cranes. This lower bound ignores side constraints. We compare the CAP, robust CAP, and a naive model where all durations are extended. To make a fair comparison with the naive approach, using *FlexC*$(\mathcal{A}, \mathcal{K})$ we minimize $\max_{a \in \mathcal{A}}(e_a + k_a)$. Table 1 shows ten instances with a time-out of 5 minutes. Entries in the first columns are the distance in percentage with the lower bound. The standard deviation on 5 runs is in parenthesis. Column α is the relative position of the robust approach compared with the initial and the naive model. Column γ is the ratio between the naive and *FlexC* results. A value of $\gamma = 2$ means the naive model doubles the distance with respect to the CAP model, compared with *FlexC*. Our approach produces a robust solution adding on average 10% to the makespan, while the naive model adds 20% on average. A fully loaded cargo would take a lower bound of (20*2*800)/4=8000 minutes to discharge, that is, 5d:13h:20m. The naive model adds 26h:40m. The robust approach adds 13h:20m, a good worst case compromise.

5 Conclusion

This paper has introduced a new declarative paradigm in order to deal with robustness in cumulative problems. We have defined a new constraint and adapted the Time-Table dynamic sweep algorithm. The experiments showed that our approach is modular, as solution performance can be preserved for a problem with many business constraints. Future work includes the adaption of other solving techniques and the use of a similar approach for other classes of optimization problems.

References

1. P. Baptiste, C. Le Pape, and W. Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.
2. N. Beldiceanu and M. Carlsson. A new multi-resource cumulatives constraint with negative heights. In P. Van Hentenryck, editor, *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2002.
3. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
4. C. Bessiere. Constraint propagation. Research report 06020 (Chapter 3 of the Handbook of Constraint Programming, F. Rossi, P. van Beek and T. Walsh eds. Elsevier 2006.), LIRMM, 2006.
5. J.-C. Billaut, A. Moukrim, and E. Sanlaville, editors. *Flexibility and Robustness in Scheduling*. Wiley, 2010.
6. Choco. 3.1.0. `URL: http://choco.sourceforge.net/`, 2013.
7. A.-J. Davenport, C. Jefflot, and J.-C. Beck. Slack-based techniques for robust schedules. In *Proc. European Conference on Planning*, pages 7–18, 2001.
8. A. Derrien, T. Petit, and S. Zampelli. Dynamic sweep filtering algorithm for FlexC. Research report RR14/1/INFO, Mines Nantes, 2014.
9. E. Hebrard. Super solutions in constraint programming. In U. Sattler, editor, *IJCAR Doctoral Programme*, volume 106 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
10. E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In J.-C. Régin and M. Rueher, editors, *CPAIOR*, volume 3011 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2004.
11. A. Lahrichi. The notions of Hump, Compulsory Part and their use in Cumulative Problems. *C.R. Acad. sc.*, t. 294:20–211, 1982.
12. A. Letort. Passage à l'échelle pour les contraintes d'ordonnancement multi-ressources. *Ph.D dissertation*, 2013.
13. A. Letort, N. Beldiceanu, and M. Carlsson. A scalable sweep algorithm for the cumulative constraint. In M. Milano, editor, *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 2012.
14. R. Leus, C. Artigues, and F. Talla Nobibon. Robust optimization for resource-constrained project scheduling with uncertain activity durations. In *Proc. IEEM*, pages 101–105, 2011.
15. L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In M. Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2004.
16. C. Wei Wu, K. N. Brown, and J. C. Beck. Scheduling with uncertain durations: Modeling beta-robust scheduling with constraints. *Computers & OR*, 36(8):2348–2356, 2009.
17. S. Zampelli, Y. Vergados, R. Van Schaeren, W. Dullaert, and B. Raa. The berth allocation and quay crane assignment problem using a cp approach. In C. Schulte, editor, *CP*, volume 8124 of *Lecture Notes in Computer Science*, pages 880–896. Springer, 2013.